

## Ejercicio 1:

- 1 a) Para un tamaño de memoria física de 4 GB y direccionamiento (espacio que ocupa una unidad de direccionamiento) de 1 B vemos que # unidades de direccionamiento  $t_b$  (callos de memoria) es  $\frac{4GB}{1B} = \frac{2^{30}B}{1B} = 4 \cdot 2^{30}$ . Luego para distinguirlas se necesita una dirección de memoria de al menos  $\log_2(2^{32}) = 32 \text{ bits} \rightarrow 12 \text{ bits}$ .
- b) Tamaño de la memoria física = 8 GB, direccionamiento = "media palabra" = 16 bits (una palabra son 32 bits) = 2B (8 bits = 1B)  $\Rightarrow$  # callos =  $\frac{8GB}{2B} = \frac{2^{30}B}{16B} = 4 \cdot 2^{26} = 2^{32}$ . Con esto, # bits =  $\log_2(2^{32}) = 32$  (la dirección de memoria debe tener al menos 32 bits).
- c) Tamaño de la memoria física = 16 GB; direccionamiento = 1 palabra = 32 bits.  $\frac{16}{8 \text{ bits}} = 4B \Rightarrow$  # callos =  $\frac{16GB}{4B} = \frac{2^{30}B}{16B} = 4 \cdot 2^{26} = 2^{32}$ . Luego # bits =  $\log_2(2^{32}) = 32$ .
- d) Tamaño de la memoria física = 32 GB; direccionamiento = "palabra doble" = 64 bits.  $\frac{16}{8 \text{ bits}} = 8B \Rightarrow$  # callos =  $\frac{32GB}{8B} = \frac{2^{30}B}{16B} = 4 \cdot 2^{26} = 2^{32}$ . Así, # bits =  $\log_2(2^{32}) = 32$ .

## Ejercicio 2:

- 2 a)iendo que la arquitectura posee palabras e instrucciones de  $y$  B (bytes), una memoria física de  $x$  B y direccionamiento = palabra ( $y$  B), luego nos que # callos =  $\frac{x}{y}B = \frac{x}{y}$  y luego la cantidad de bits necesarios para distinguirlas en una dirección es  $\lceil \log_2(\frac{x}{y}) \rceil = \lceil \log_2(x) - \log_2(y) \rceil$  (donde  $\lceil x \rceil = \min\{k \in \mathbb{Z}, k \geq x\}$ ).  
 Nótese que no tomamos en cuenta el caso en que # callos  $\leq 1$  ya que en dicha circunstancia la memoria no tendría callos en los cuales almacenar datos.
- b)iendo que las instrucciones son de  $y$  B =  $8 \cdot y$  bits y cada una posee un operando con modo de direccionamiento a memoria, esto nos dice que de los  $8 \cdot y$  bits se usan  $\lceil \log_2(x) - \log_2(y) \rceil$  bits para una dirección de memoria. Esto nos deja con  $8 \cdot y - \lceil \log_2(x) - \log_2(y) \rceil$  bits disponibles para el código de operación. Como los bits sólo pueden tomar 0 o 1, el número máximo de códigos de operación es  $2^{8 \cdot y - \lceil \log_2(x) - \log_2(y) \rceil}$ .
- c) Si ahora  $x = 2^k$  y además  $y = 2^j$  donde  $k, j \in \mathbb{N}$  ( $x$  y  $y$  son potencias de dos), vemos que ahora la cantidad de bits necesarios para distinguir todos los callos de la memoria es  $\lceil \log_2(2^k) - \log_2(2^j) \rceil = \lceil k - j \rceil = k - j$  (pues  $k, j \in \mathbb{N}$ ). A su vez, la máxima cantidad posible de códigos de operación es  $2^{8 \cdot 2^j - (k-j)} = 2^{8 \cdot 2^j - k + j}$ .
- d) La máxima granularidad de acceso a memoria se obtiene cuando la unidad direccionable es la más chica posible. Al ver el direccionamiento de la arquitectura a palabras y una palabra tiene 2 bytes, para ser la más chica posible y debe ser 0, así la unidad direccionable es de  $2^0 = 1$  byte, lo cual hace que para un tamaño de memoria dado la cantidad de callos sea máxima.

## Ejercicio 3:

- 3 a)iendo que las instrucciones son de 12 bits, los direccionamientos de 4 bits, y debe haber 6 instrucciones con dos direcciones, primero tomamos que para las últimas 6 bits reservamos 6 bits (espacio equivalente a dos direcciones) de los 12 bits que los forman, y con los otros 4 ( $12 - 8 = 4$ ) bits tomamos 6 combinaciones de 0 y 1. Las diferentes para denotar a cada instrucción. Nótese que en 4 bits hay  $2^4 = 16$  combinaciones posibles, de las cuales usamos 6 para los de dos direcciones y restamos 10 para el resto. Si queremos que la cantidad sea máxima, para cada una de las combinaciones nos vamos a reservar 4 bits de los nodos anteriormente para los direccionamientos y tendremos los otros 4 disponibles para variar códigos de operación. En 4 bits se tienen  $2^4 = 16$  combinaciones y como esto es por cada una de las 10 combinaciones que no denotan a los de dos direcciones, se tienen por lo tanto del producto:  $10 \cdot 2^4 = 160$  instrucciones como máximo de una dirección.



b) En este caso, las instrucciones son de 16 bits, las direcciones de 6 bits y hay  $n$  instrucciones de dos direcciones. Para los últimos reservo 12 bits de los 16 disponibles para las direcciones, y uso los otros 4 bits para hacer  $n$  diferentes códigos de operación (notese que como  $2^4 = 16$ ,  $n$  no puede ser mayor a 16) pues sino la cantidad de instrucciones valdría cero. Ahora, en los  $2^4 - n = 16 - n$  combinaciones restantes de dichos 4 bits, reservo 6 bits de los otros 12 bits para la dirección y tengo 6 bits restantes para el código de operación. Para 6 bits hay  $2^6$  combinaciones, quedando  $(16 - n) \cdot 2^6$  instrucciones de una dirección.

## Ejercicio 4:

4) Longitud de las instrucciones = 16 bits, Longitud de la dirección de memoria = 4 bits. En esto voy que primero debo contemplar 15 instrucciones de 3 direcciones para lo cual reservo  $12 - 4 = 12$  bits para las direcciones y con los  $16 - 12 = 4$  restantes tomo 15 combinaciones para denotar a dichas instrucciones.iendo que  $2^4 = 16$  me queda  $16 - 15 = 1$  combinación de dichos 4 bits para el resto de las instrucciones. Luego, voy que debo tener 14 instrucciones de 2 direcciones, para las cuales reservo  $2 \cdot 4 = 8$  bits de 12 disponibles (teniendo en cuenta que los otros 4 poseen una combinación que los distingue de los primeros 15 instrucciones). Luego, voy que los  $4 - 12 = 8$  bits no especificados debo usarlos para formar 14 códigos de operación diferentes, quedando  $2^4 - 14 = 16 - 14 = 2$  combinaciones posibles para dichos 4 bits en el resto de las instrucciones. Seguido a esto, tengo 31 instrucciones de 1 dirección, para las cuales reservo 4 bits de los 8 bits no usados para los códigos de operación de las instrucciones anteriores. Así, me quedan 4 bits para realizar 31 códigos de operación, aunque por los 2 combinaciones restantes de las instrucciones de dos direcciones voy que tengo 4 bits por cada una de las restantes dichas y así  $2 \cdot 2^4 = 2^5 = 32$  combinaciones disponibles. De ellos, tomo 31 dejando 1 posible combinación restante para dichos 4 bits que permita marcar los últimos 4 bits. Juntamente, al resto contemplar 16 instrucciones sin direcciones y 4 bits libres para una única combinación del resto, voy que  $2^4 = 16$ , con lo cual uso todas mis combinaciones posibles para hacer los códigos de operación de dichas instrucciones. De esta forma se tiene el formato general de instrucción que contemple lo dicho. Un caso particular sería:

- Instrucciones de 3 direcciones:  $[ZZZZ XXXX XXXX XXXX]$  donde ZZZZ va de 0001 a 1111 (15) y XXXX son los espacios reservados para las direcciones de memoria. No se usó ZZZZ = 0000.
- Instrucciones de 2 direcciones:  $[0000 ZZZZ XXXX XXXX]$  donde ZZZZ va de 0010 a 1111 (2 a 15 = 14) y XXXX son los espacios reservados para las direcciones de memoria. No se usaron ZZZZ = 0000, 0001.
- Instrucciones de 1 dirección:  $[0000 0000 ZZZZ XXXX]$  donde ZZZZ va de 0001 a 1111 (1 a 15 = 16) y XXXX es el espacio reservado para la dirección de memoria. No se usó ZZZZ = 0000.
- Instrucciones sin direcciones:  $[0000 0000 0000 ZZZZ]$  donde ZZZZ va de 0001 a 1111 (1 a 15 = 16), representando los códigos de operación de todas las 16 instrucciones.

## Ejercicio 5:

5) Código de operación: extensible; longitud de la instrucción: 36 bits. Con estos datos, propóngame el siguiente formato de instrucción:

1. Instrucciones con dos direcciones de 15 bits y un registro de 3 bits:  $[RRR\ RRR\ XXX\ XXX\ XXX\ XXX\ XXX\ XXX\ XXX\ XXX]$  donde RRR es el espacio reservado para el registro,  $XX\ XXX\ XXX\ XXX\ X$  y  $YYY\ YYY\ YYY\ YYY$  son los espacios reservados para las dos direcciones de memoria y AAA va de 001 a 111, contemplando 7 combinaciones correspondientes a sus códigos de operación. No se usó AAA = 000.

2. Instrucciones con una dirección de 15 bits y un registro de 3 bits:  $[000\ AAA\ AAA\ AAA\ 00R\ RRR\ XXX\ XXX\ XXX\ XXX]$  donde RRR es el espacio reservado para el registro,  $XXX\ XXX\ XXX\ XXX$  es el espacio reservado para la dirección y  $AAA\ AAA\ AAA$  va de 000 0000 0000 a 0001 1111 0011 (de 0 a 494+500) contemplando los códigos de operación de las 500 instrucciones dadas.

3. Instrucciones sin direcciones ni registros:  $[0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 00AA\ AAAA]$  donde AA-AAAA va de 00 0000 a 11 0001 (0 a 49+50) contemplando los códigos de operación de las 50 instrucciones dadas.

## Ejercicio 6:

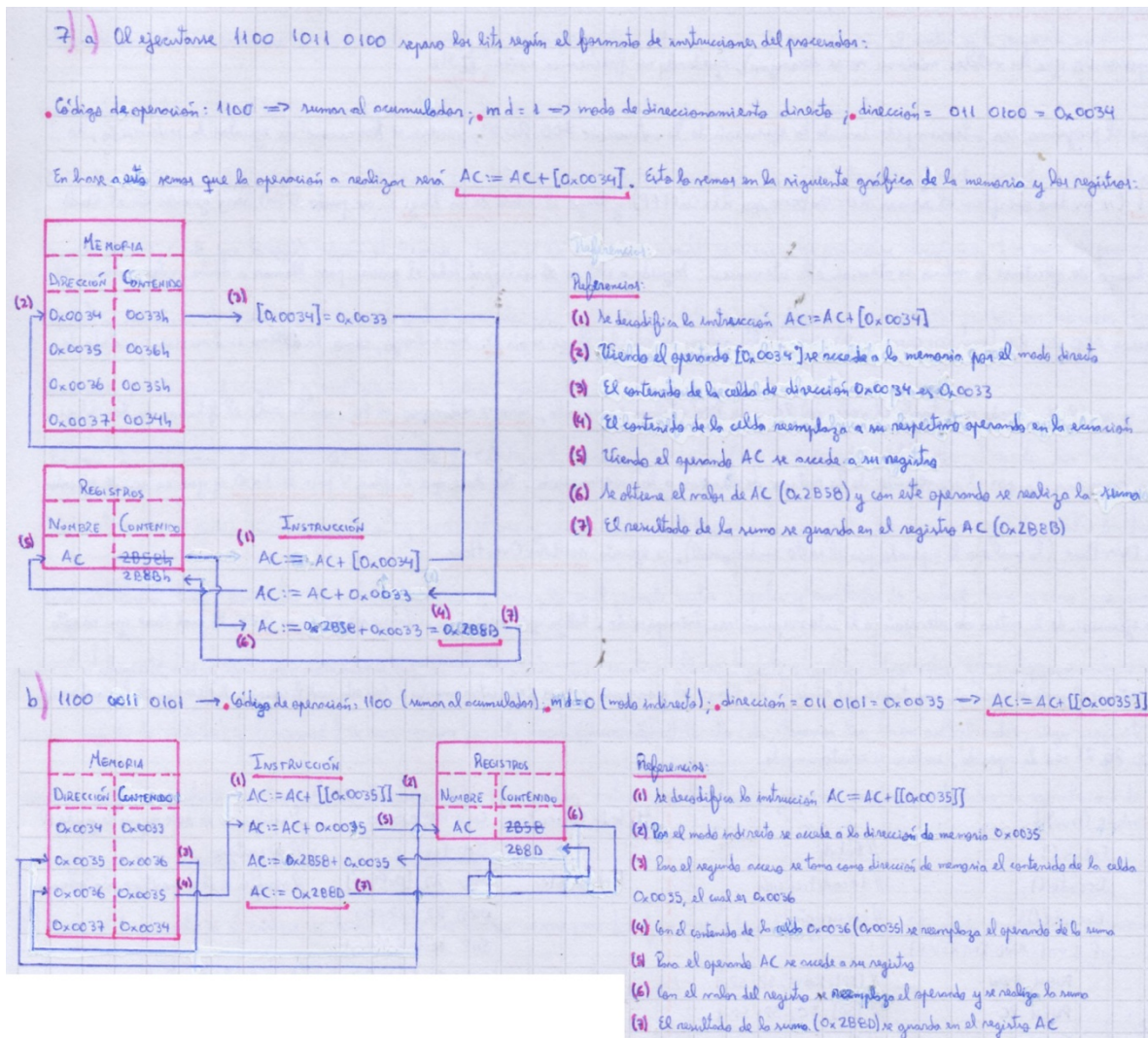
6) Dirección del registro = 3 bits; longitud de la instrucción = 12 bits. Suponemos que diseñar dicho formato de instrucción es posible. Con eso, vemos que para las 4 instrucciones con 3 registros necesitamos 9 bits para los mismos, dejando 3 bits para 4 códigos de operación. Notemos que  $2^3 = 8 > 2^2 = 4$ , por lo que para designar a dichos códigos podemos dejar un bit fijo y variar los otros dos. A dicho bit lo llamaremos X.

Segundo a esto debe contemplar las 255 instrucciones con un registro. Para diferenciarlas de las anteriores, cambia el valor del bit X y eso que tengo 11 bits disponibles, de los cuales reservo 3 para el registro, dejando 8 para los códigos de operación. Viendo que  $2^8 = 256$  y debe contemplar 255 instrucciones, repara una combinación de dichos 8 bits y uno los restantes para los códigos de operación de estas instrucciones.

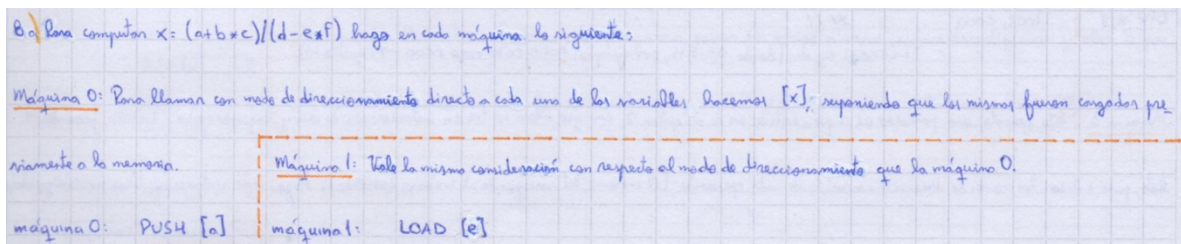
Notemos que hasta ahora las únicas combinaciones no utilizadas para denotar una instrucción son las que poseen el bit X con el mismo valor que el de las instrucciones de un registro y los 8 bits en la combinación no usada para denotar su código de operación. Esto deja 3 bits disponibles y en total  $2^3 = 8$  combinaciones. Sin embargo, nos piden contemplar 16 instrucciones más, y  $16 > 8$ , dejando algunos de ellos sin satisfacer y generando un abuso. Finalmente, dicho diseño de formato de instrucción es irrealizable.



## Ejercicio 7:



## Ejercicio 8:





PUSH [b]	MUL [F]	Máquina 2 & 3: Para estas tomamos a una variable X como X si es un negativo o [X] si se almacena en la memoria. Comenzamos la variable auxiliar Y para hacer los cálculos.	
PUSH [c]	STORE [x]		
MUL	LOAD [i]		
ADD	SUB [x]		
POP [x]	STORE [x]	máquina 2:	Mov x, e
PUSH [e]	LOAD [c]		Mov x, f
PUSH [f]	MUL [b]		Mov y, d
MUL	ADD [a]		SUB y, x
PUSH [d]	DIV [x]		MOV x, c
SUB	STORE [x]		MUL x, b
PUSH [x]			ADD x, a
DIV			DIV x, y
POP [x]			MOV x, y
		máquina 3:	MUL y, b, c
			ADD y, a, y
			MUL x, e, f
			SUB x, d, x
			DIV x, y, x

b) Podemos ver en base a lo hecho en a) que la cantidad de instrucciones de cada programa (para cada máquina) son: 0: 14, 1: 11, 2: 8, 3: 5.

c) Máquina 0: Siendo el código de operación de 8 bits, todos los modos de direccionamiento directos y 6 instrucciones en total, podemos hacer:

Instrucción	Código de operación	Dirección (opcional)	Notas que tomamos la dirección de memoria opcional pues sólo se usa en las primeras dos instrucciones. Siendo que [M] ocupa 16 bits y el código de operación 8 bits, las instrucciones tienen longitud múltiplo de 4 bits. Así queda el formato como:
PUSH [M]	0000 0001	[M]	
POP [M]	0000 0010	[M]	
ADD	0000 0100	—	
SUB	0000 1000	—	
MUL	0001 0000	—	
DIV	0010 0000	—	

Máquina 1: Al igual que la anterior, todos los modos de direccionamiento coinciden y sólo hay 6 instrucciones para un código de operación de 8 bits.

Instrucción	Código de operación	Dirección de memoria	Siendo que [M] ocupa 16 bits y el código de operación es de 8 bits, todas las instrucciones son de 24 bits, que es múltiplo de 4 bits. Con esto, queda el formato de instrucción como:
LOAD [M]	0100 0001	[M]	
STORE [M]	0100 0010	[M]	
ADD [M]	0100 0100	[M]	
SUB [M]	0100 1000	[M]	
MUL [M]	0101 0000	[M]	
DIV [M]	0110 0000	[M]	

Máquina 2: Al disponer de 16 registros e instrucciones de dos operandos, necesitamos unos 4 bits adicionales de los cuales los primeros 2 corresponden al primer operando y los otros 2 al otro (con 4 debido a que la longitud de la instrucción debe ser múltiplo de 4 bits). Luego, asigno a cada instrucción un código de operación:

Instrucción	Código de operación	Con esto tenemos que la instrucción posee el formato: Código de op. (8 bits) Modo de dir. (4 bits) Operando 1 (8 bits) Operando 2 (8 bits)
MOV X, Y	1000 0001	donde Modo de dir = XX YY siendo XX, YY = 00 (si el modo de direccionamiento del operando es directo), 11 (si es un registro); los operandos poseen la dirección de memoria si el modo es directo, o el i-ésimo bit en 1 y el resto en 0 para indicar que se refiere al i-ésimo registro (donde 0 ≤ i ≤ 15, por ejemplo: 0000 0010 0000 0000 → registro 10).
ADD X, Y	1000 0010	
SUB X, Y	1000 0100	
MUL X, Y	1000 1000	
DIV X, Y	1001 0000	

Máquina 3: Este formato será parecido al implementado en la máquina 2, aunque ahora se tienen instrucciones de dos y tres operandos. Por ello, tomamos 4 bits para indicar los modos de direccionamiento de cada operando (el i-ésimo bit corresponde al i-ésimo operando). Luego, cada instrucción tiene un código dado:







## Ejercicio 9:

9 a. Tamaño de la memoria física = 2GB; Registros = 624 ( $2^9$ ) de 64 bits divididos en 4 partes de 16 bits; Tamaño de la unidad direccionable: 1B  
 Para diseñar un formato de instrucciones de longitud fija de 64 bits, sea primero que # unidades direccionables =  $\frac{2GB}{1B} = \frac{2^{30}}{2^8} = 2^{22} \Rightarrow$  la cantidad de bits de la dirección de memoria equivale a  $\log_2(2^{22}) = 22$ . A su vez, como hay  $2^9$  registros, los pudo distinguir con una cadena de  $\log_2(2^9) = 9$  bits.  
 Ahora sea que la instrucción MOV p, q es la que requiere la mayor reserva al tener que reservarse en un caso 62 bits correspondientes a las direcciones de memoria. Esto deja 2 bits libres para el código de operación y el modo de direccionamiento. Como a los bits como 11 para indicar que la operación es MOV con modo de direccionamiento directo en ambos operandos. Nótese que para los otros modos de direccionamiento el espacio a reservarse es menor ya que el espacio reservado para una de las direcciones de memoria (al menos) pasará de 31 a 10 bits, por lo que tengo la combinación 01 para indicar que la instrucción es MOV y los operandos son dirección de memoria y registro o ambos registros. Siendo que ahora tengo 21 bits disponibles para dos combinaciones y distinguir entre estos 3 casos, sea que puedo tomar solo los primeros 2 para distinguirlos y dejar el resto en 0. Si el primer operando es una dirección toma 10, si es el segundo 01 y si ambos son registros 00. De esa forma, para MOV p, q tengo:

1 Dirección de memoria (31 bits) 1 Dirección de memoria (31 bits)	→ MOV [p], [q]	Nótese que usamos 0 para indicar que una cantidad con
0 Dirección de memoria (31 bits) 1 10 0 (19 bits) Registro (10 bits)	→ MOV [p], Rq	reservar de bits es nulo (todos ceros). Otro para el resto
0 Dirección de memoria (31 bits) 1 01 0 (19 bits) Registro (10 bits)	→ MOV Rp, [q]	
0 0 (5 bits) Registro (10 bits) 0 (6 bits) Registro (10 bits) 1 0 (31 bits)	→ MOV Rp, Rq	

Las instrucciones son las combinaciones que este formato me da

para pasarlas. Después luego tengo las instrucciones (INT R<sub>u</sub>(i), R<sub>v</sub>, R<sub>w</sub> y SDV R<sub>u</sub>, R<sub>v</sub>, R<sub>w</sub>(i)) que tienen que reservarse 32 bits provenientes de los números de los 3 registros y una parte del primero o el tercero según la instrucción. Para estos reservo los bits 32-61 para los números de registro y de los bits 31 y 63 en 0 para distinguir a estas operaciones de MOV (contando de 0 a 63 desde el menos significativo). Esto deja al bit 62 disponible para distinguir una instrucción de la otra, y de los 31 bits menos significativos reservo los bits 30 y 29 para escribir el número de parte del registro, dejando el resto en caso:

0 0 Registro (10 bits) Registro (10 bits) Registro (10 bits) 0 Parte del Registro (2 bits) 0 (29 bits)	→ INT R <sub>u</sub> (i), R <sub>v</sub> , R <sub>w</sub>
0 1 Registro (10 bits) Registro (10 bits) Registro (10 bits) 0 Parte del Registro (2 bits) 0 (29 bits)	→ SDV R <sub>u</sub> , R <sub>v</sub> , R <sub>w</sub> (i)

Seguido a esto tengo las instrucciones VEC y ADD que toman 3 registros para lo cual debo reservarse 30 bits. Para distinguirlos de los anteriores tomamos los bits 31 y 63 en 0 (no es MOV) y el bit 28 en 1 (no es ninguna de las 2 anteriores). Luego, reservo los bits 32-61 para los registros y el bit 62 para distinguir entre una y la otra. El resto quedarán en 0:

0 0 Registro (10 bits) Registro (10 bits) Registro (10 bits) 0 00 1 0 (28 bits)	→ VEC R <sub>u</sub> , R <sub>v</sub> , R <sub>w</sub>
0 1 Registro (10 bits) Registro (10 bits) Registro (10 bits) 0 00 1 0 (28 bits)	→ ADD R <sub>u</sub> , R <sub>v</sub> , R <sub>w</sub>

Por último tomamos la instrucción CLR R<sub>u</sub>(i) para la cual debemos reservarse 12 bits correspondientes al número de registro y la parte del mismo. Para distinguir de los anteriores tomaremos los bits 31, 62 y 63 en caso, los bits 28-30 en caso y el bit 27 en 1. Luego reservo los bits 52-61 para el número de registro y los 2 anteriores para la parte (el resto en 0), dejando:

0 0 Registro (10 bits) Parte i (2 bits) 0 (16 bits) 00 00 1 0 (27 bits)	→ CLR R <sub>u</sub> (i)
---	--------------------------



- b) Este formato emplea los modos de direccionamiento directo (en la instrucción MOV) y registro (en la misma y en el resto, más allá de que a veces no es su valor completo sino una parte).
- c) La longitud de palabra más adecuada sería de 32 bits, puesto que permite decodificar las instrucciones con sólo dos accesos a la memoria. Para el modo de direccionamiento directo emplea sólo un acceso para ir a una dirección dada y para tomar el número de todos los registros que son operandos de las instrucciones de dos a tres en un sólo acceso.iendo que la velocidad del bus suele ser el cuello de botella en la velocidad del procesamiento y funcionamiento de la máquina, una menor cantidad de accesos aunque más compleja la hacen más rápida.

## Ejercicio 10:

- 10 a)iendo que los direccionales son de 8 bits, y que el direccionamiento es a palabra, lo cual es de 8 bits, eso que en una cadena de 8 bits puedo hacer  $2^8$  combinaciones (al ser cada bit 0 o 1) lo cual corresponde a la mayor cantidad de unidades direccionables. Con esto, eso que el tamaño máximo de memoria física equivale al producto entre la cantidad de unidades direccionables y su tamaño, quedando  $2^8 \times 8 \text{ b} = \frac{16}{8} = 2^8 \text{ B}$
- b) El tamaño del stack pointer debe ser de 8 bits, pues al estar el stack representado dentro de la memoria, este registro debe contener la dirección de memoria (de tamaño 8 bits) de la celda en la que pueden almacenarse nuevos elementos o la misma.
- c) Para el diseño del formato eso que las instrucciones deben ser de longitud fija, todas las instrucciones tienen un modo de direccionamiento definido, y hay 4 registros que se pueden distinguir con  $\log_2(4) = 2 \text{ bits}$ . Con ello, eso las instrucciones ADD, JG y SWAP los cuales requieren que reserve 4 b correspondiente a dos operandos de registros en cada uno (son los que requieren mayor espacio a reserven). Así vez, eso que debo distinguir a los 3 con códigos de operación y ahí una cadena de 4 bits es suficiente pues  $2^4 = 16 > 3$ . Esto deja 13 combinaciones disponibles de dichos 4 para el código de operación del resto de las operaciones (son 3 pues la longitud de las instrucciones es de una palabra que es de 8 b). Comencemos los bits más significativos para el código de operación. Luego tomamos las operaciones LCR4 y LCR4 que deben reserven 4 bits para la constante, y para los 4 bits siguientes tienen 13 combinaciones disponibles para distinguir su código de los anteriores (dejando 11). Luego, para las 4 instrucciones rotatas debo reserven 2 bits para numerar el registro y para los 4 bits siguientes tengo 11 combinaciones disponibles, y a su vez por cada una tengo todas las combinaciones de los dos rotatas pues los primeros 4 ya los distinguí de los anteriores. Esto me deja  $11 \cdot 2^2 = 44$  combinaciones disponibles para distinguir 4 códigos, lo cual es más que suficiente y al diseño terminé. Ahora que ahora tomamos un diseño particular de todos los complementables:

Instrucción	Formato	Instrucción	Formato
ADD R <sub>i</sub> R <sub>j</sub>	0000 Registro (2b) Registro (2b)	STORE R <sub>i</sub>	0101 00 Registro (2b)
JG R <sub>i</sub> R <sub>j</sub>	0001 Registro (2b) Registro (2b)	LOAD R <sub>i</sub>	0101 01 Registro (2b)
SWAP R <sub>i</sub> R <sub>j</sub>	0010 Registro (2b) Registro (2b)	ZERO R <sub>i</sub>	0101 10 Registro (2b)
LCR4 cte4	0011 Constante de 4 bits	NOT R <sub>i</sub>	0101 11 Registro (2b)
LCR4 cte4	0100 Constante de 4 bits		



d) viendo que el formato de instrucciones es de longitud fija de una palabra (8b) pero que si se pudieran agregar instrucciones, estas deberían tener un código de operación que los distinga del resto. Para los 4 bits superiores pero que tengo las combinaciones 0000, 0001, 0010, 0011 y

0100 ocupados por las instrucciones de los registros y los de la constante de 4 bits. Además, la combinación 0101 también está ocupada, pues las 4 instrucciones de un registro agotan todas las combinaciones de los bits 3 y 4, y de las 3 menos significativas. Esto deja 10 combinaciones disponibles para las primeras (las más significativas) 4, y para cada uno de ellos se tienen todas las combinaciones de los últimos (menos significativos) 4, o sea,  $2^4 = 16$ . viendo que las instrucciones no poseen operandos, podemos tomar a todos ellos como códigos de operación, distinguiendo (por la regla del producto) así:  $10 \times 16 = 160$ .

10e) viendo que no se modifica el diseño, no puedo agregar LCTE de B cambiando el formato de instrucción, pero puedo cambiar el programa ensamblador para que cuando sea que se quiere ejecutar dicha instrucción, esta se simule tomando los 4 bits más significativos de la constante y haciendo LCTEH con dicha porción, y a su vez tomando los 4 restantes (los menos significativos) y haciendo LCTEL. De esa forma, la constante de B para el registro R0 (suponen la constante de B en dos constantes de 4b no equivale a ejecutar una microinstrucción, sino a modificar el programa para que sea una parte primero y otra después).